Engenharia Informática e Computação

Agentes e Inteligência Artificial Distribuída

Sociedade de agentes baseados em arquitecturas BDI

Relatório Final

Versão 1.0

Carlos Figueiredo, ei99030@fe.up.pt
José Luís Silva, ei99016@fe.up.pt
Vítor Pinto, ei99036@fe.up.pt
(Turma 1)

13 de Dezembro de 2002

Índice Geral

1 Introdução

Nesta secção faz-se uma introdução ao trabalho, aos seus objectivos ao seu enquadramento no âmbito da disciplina e à motivação que levou à sua implementação.

1.1 Enquadramento

O presente documento desenvolvido no âmbito da cadeira de AIAD, pretende implementar um sistema multi-agente. Este é um dos temas de maior relevo discutidos nesta disciplina pois trata a criação e a definição de agentes para alem disso este trabalho também implica a interacção entre agentes. É feita ainda a descrição dos esquemas de representação de conhecimento noutra secção. Os detalhes de implementação mais importantes são descritos no módulo seguinte. Em anexo é incluído um manual de utilizador e é dado um exemplo de execução.

1.2 Motivação

O maior interesse na realização deste trabalho prende-se com a criação de um trabalho multi-agente e na possibilidade de implementar mecanismos que os tornem inteligentes.

1.3 Objectivos

Este trabalho pretende implementar um conjunto de agentes baseados na arquitectura BDI[1]. O sistema deve utilizar a plataforma JADE [3] para suporte ao desenvolvimento dos agentes. Devem existir três tipos de agentes diferentes e o sistema deve possibilitar que estes tipos de agentes coabitem. O sistema deve disponibilizar uma interface gráfica, construída usando a linguagem Java, que possibilite ao utilizador definir o mundo onde os agentes habitaram e onde seja possível visualizar as acções dos agentes.

[1] Ver glossário

Deve ser possível também através da interface gráfica poder especificar as propriedades e criar agentes.

1.4 Estrutura do Documento

Este relatório está dividido em secções, em que a primeira se dedica à descrição do problema. Seguidamente apresenta-se as funcionalidades implementadas. O ponto seguinte especifica a divisão em módulos do programa utilizado, sendo aqui descrito cada um dos módulos implementados.

2 Descrição do Problema (implementado)

Este trabalho pretende fazer uma implementação de agentes, definidos segundo a arquitectura BDI. Os agentes devem poder deslocar-se por um conjunto de caminhos que constituem um grafo. Existe também uma fonte que serve como objectivo aos agentes bons. O conjunto de caminhos e fonte constitui o mundo, este deve ser disponibilizado pelo utilizador. Neste sistema existem três tipos diferentes de agentes, os agentes monstro, os agentes energia e os agentes bons. De seguida é feita a sua descrição.

***Agente monstro*:** Este agente deve detectar a existência de agentes bons e de agentes energia dentro do seu campo de visão. Este valor é pré-especificado. Este agente deve detectar quais os caminhos mais próximos para onde se pode movimentarem. A sua movimentação deve ter em conta a sua localização actual, orientação e as ultimas posições dos agentes bons. Existe um valor de "cólera" que aumenta quando este agente consegue visualizar algum agente bom e decrementa quando este não se encontra no seu campo de visão. Existe outro valor associado a este agente que é a "alegria", este relaciona-se directamente com a distância aos agentes bons. O agente tem um valor de "energia" que decresce quando este se movimenta e aumenta quando vê um agente energia. O valor de energia é reposto quando o agente entra em contacto com um agente energia. O objectivo deste agente é maximizar a sua "alegria" e minimizar a sua "cólera",

mantendo sempre valores para a sua “energia” aceitáveis, pois caso a sua “energia” seja negativa o agente morre.

***Agente "bom"*:** O objectivo principal deste tipo de agentes é alcançar a fonte. Dentro do seu campo de visão este agente é capaz de detectar os agentes monstro e os agentes energia, para alem da fonte. Deve ainda detectar os caminhos mais próximos para onde se deve movimentar tendo em consideração a sua posição actual, orientação e as ultimas posições conhecidas do agente monstro. Estes agentes têm um valor de “medo” associado que é proporcional à distância ao agente monstro. Existe também um valor de “alegria” que varia de forma oposta ao seu “medo”. Este agente também tem um valor de “energia” que é em tudo idêntico ao valor correspondente do agente monstro. Este agente deseja maximizar a sua “alegria” e minimizar o “medo”.

***Agente energia*:** Este agente têm conhecimento dos últimos caminhos percorridos e é capaz de detectar quais os caminhos mais próximos para se movimentar. Tem um valor associado de “alegria” que aumenta quando este consegue incrementar o valor de energia dos outros agentes.

3 Funcionalidades

A nossa aplicação implementa várias funcionalidades. Sendo uma sociedade de agentes, o utilizador que execute a aplicação terá a hipótese de criar agentes e o mundo onde eles se movimentam.

Em relação à criação de agentes, o utilizador tem à sua disposição, três tipos de agentes: “Monstro”, “Bom” e “Energia”.

Ao criar um agente do tipo “Monstro”, o utilizador poderá definir para esse agente características comuns a todos os agentes como a posição que este vai ter no mundo, o seu raio de visão e o valor da sua alegria inicial. Além destas características comuns a todos os agentes, o agente monstro tem a particularidade de poder ser definido para ele um valor inicial de cólera e um coeficiente de “aventureiro” que é utilizado para

especificar o comportamento do agente face a determinadas situações de decisão. Para este agente pode ser definida a quantidade de energia inicial que ele possui.

Ao criar um agente do tipo "Bom", o utilizador poderá definir todas as características comuns a todos agentes, que foram descritas no parágrafo acima, assim como a característica específica dos agentes "Bom" que é o valor do seu medo inicial e um factor de medo que, como no agente "Monstro", influencia a próxima acção do agente. O valor inicial de energia deste agente pode também ser definida.

Um agente do tipo "Energia", apenas tem as características básicas comuns a todos os agentes com a pequena excepção de que a sua energia é ilimitada não podendo ser definido o valor inicial da sua energia.

Em relação à criação do mundo, o utilizador poderá definir caminhos (através de nós) por onde os agentes se vão movimentar. Depois de se ter o grafo construído, o utilizador poderá colocar nos nós, vários agentes dos três tipos definidos acima, sendo ainda obrigado a colocar num dos nós uma e uma só "fonte" sem a qual o programa não funcionaria. Depois de criado o mundo, o utilizador poderá ainda salvar o mesmo podendo depois carregá-lo noutra ocasião.

Tendo o mundo e os agentes criados o programa é automático executando, os agentes, as suas tarefas. O utilizador pode ainda fazer uma pausa na execução podendo depois esta continuar.

4 Estrutura do Programa

Esta secção pretende introduzir a estrutura do trabalho, através da descrição dos módulos que constituem os trabalhos.

Como é visível na figura 1, este trabalho é constituído por seis módulos, estes são: Ontologia; Agentes; GUI, Behaviors; Mundo, que foram desenvolvidos. É usada a plataforma JADE, representada na figura 1, pela classe Jade. De seguida é feita uma descrição dos módulos.
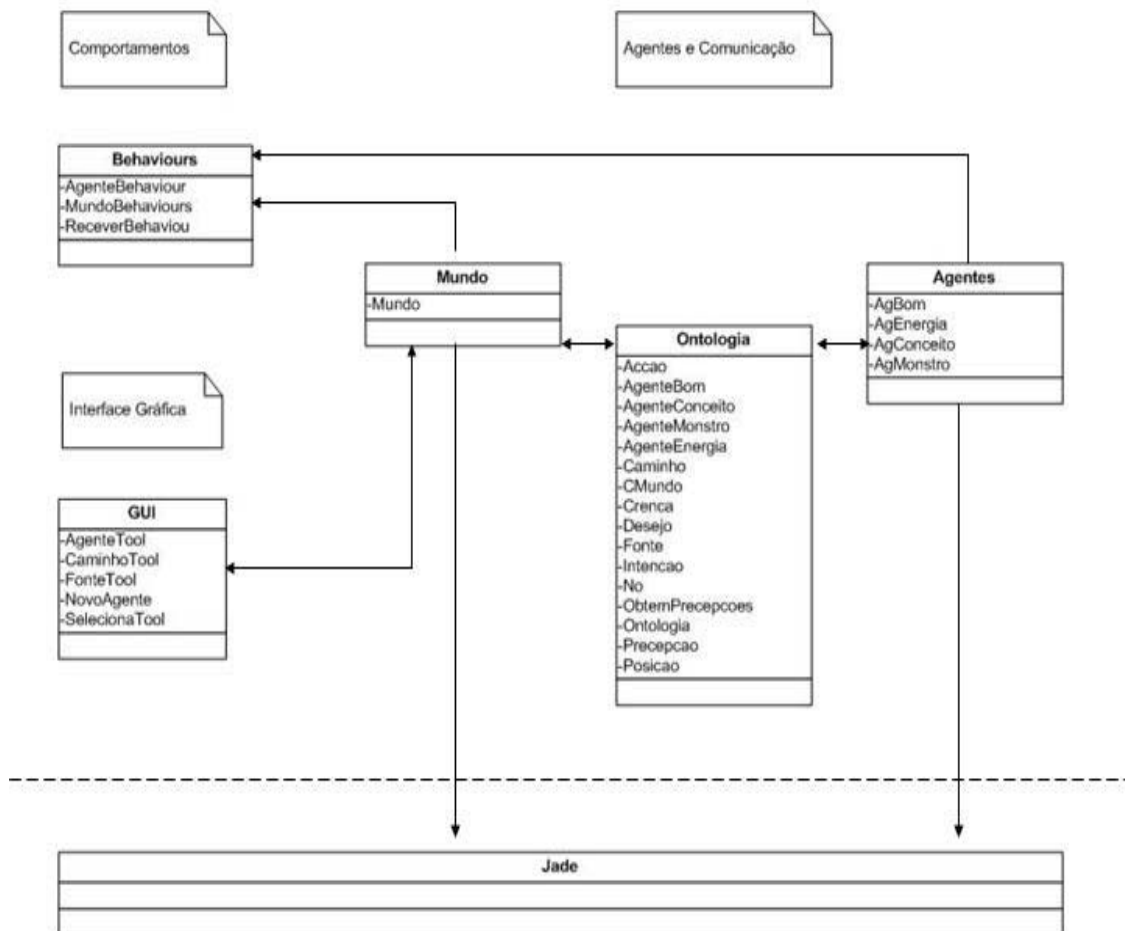
Figura 1 Diagrama de módulos do programa.

4.1 Módulo Ontologia

Este método pretende implementar um conjunto de conceitos usados pelo programa, definindo desta forma uma Ontologia, de acordo com [4]. Estes termos são usados pelos restantes módulos, nomeadamente na troca de mensagens. De seguida é feita uma pequena descrição das classes que constituem este componente, sendo dado maior relevo aos termos mais relevantes para o trabalho.

- **Accao** Representa uma acção que um Agente pode ter. Essa acção é identificada por uma string que a especifica e por um No que representa o local ao qual esta associado a acção. Por exemplo, se estivermos perante uma acção que faça o agente ficar parado, este No representa a posição actual do Agente.

- **Agente Conceito** Esta classe especifica as propriedades comuns dos três tipos de agentes existentes. Algumas dessas propriedades são: o seu Nome, A sua Alegria, o Mundo ao qual pertencem, o seu factor de "Aventureiro" e o seu raio de visão.

- **Agente Bom Agente Monstro e Agente Energia** Estas três classes complementam (Estendem) a classe anterior, de forma a especificar as características individuais de cada tipo de agente. Algumas dessas características individuais são: Medo (Bom), Energia (Bom e Monstro) e Cólera (Monstro).

- Caminho Esta classe implementa um caminho desenhado pelo utilizador. Ao nível da sua estrutura um caminho consiste em dois Nós (origem e destino). Esta classe tem ainda um valor de Peso que representa a distância entre os dois Nós. Esta distância é utilizada para cálculos de actualização de Energia, quando o agente atravessa o caminho.

- **Cmundo** Esta classe especifica as propriedades de um mundo, nomeadamente a posição da Fonte e os Caminhos que o constituem.

- **Crenca** Esta classe implementa o conceito de uma Crença do agente. Uma Crença representa algo que um agente viu no seu raio de visão. No ponto seguinte este conceito será analisado com maior pormenor.

- **Desejo** Esta classe representa os Desejos que os Agentes podem ter com base naquilo que em certa altura têm nas suas Crenças. Também este conceito será analisado com maior pormenor seguidamente.

- **Fonte** Esta classe representa o conceito de fonte, basicamente esta representação é apenas feita pelo Nó onde a fonte se situa.

- **Intencao** Esta classe representa as Intenções que um determinado Agente pode ter em determinada altura. Também este conceito será detalhado mais à frente.

- **No** Um No representa uma posição no mundo, esta é constituída por um par de coordenadas x, y.

- **ObtemPrecepcoes** Esta classe implementa o conceito de uma função que permite ao Agente pedir ao Mundo as percepções que está a ver. Basicamente este conceito é constituído por um Agente e por uma Percepção.

- **Ontologia** Esta classe permite construir a Ontologia, de acordo com [4] e registando na ontologia os conceitos que constituem este módulo.

- **Precepcao** Esta classe define uma Percepção que representa aquilo que o agente vê em determinada altura.

- **Posicao** Uma posição é uma estrutura de dados usada para a representação que o agente faz do mundo. Numa fase posterior retornaremos a este tema.

4.2 Módulo Agentes

Este módulo implementa os agentes BDI, logo é constituído por quatro classes: AgBom, AgEnergia, AgMonstro e Agente. A classe Agente define propriedades comuns aos três tipos de agentes BDI. Estas classes serão detalhadas seguidamente.

- **Agente** Esta classe especifica um conjunto de funcionalidades e características comuns aos três tipos de agentes. Algumas dessas características são os

tipos de Crenças e de Acções, pois estas são comuns a todos os agentes. É ainda especificada a escala de Energia que os vários agentes podem ter.

As funcionalidades que aqui são disponibilizadas, permitem aos agentes construir e actualizar as suas Crenças, permitem ainda manter e actualizar a memória do mundo que um Agente já viu, relembre-se aqui o facto de que um agente tem um raio de acção que lhe permite ver apenas parcialmente o mundo, desta forma progressivamente é construída uma representação do mundo na memória do Agente. Outra funcionalidade aqui implementada permite encontrar todos os Caminhos existentes entre duas posições do mundo, podendo assim o agente saber quais todas as hipóteses que tem em determinado momento para alcançar certo objectivo. Existe ainda a funcionalidade que permite a um agente saber quais são todas as acções que em certa altura pode tomar.

- **AgBom, AgEnergia e AgMonstro** Estas três classes complementam a classe Agente, definindo-se desta forma as características e funcionalidades de cada um dos tipos de agentes. Como todos os agentes têm a mesma estrutura, as funcionalidades são as mesmas, deferindo na sua implementação. Essas funcionalidades permitem a um Agente obter os seus Desejos e as suas Intenções. Permitem também seleccionar uma Acção e executá-la.

Estas classes definem também as propriedades individuais de cada Agente, como os seus Desejos e o tipo de Intenções que lhe estão associados. Definem ainda a escala de medo (Bom) e a escala de cólera (Monstro). Estas classes implementam ainda as funcionalidades que permitem registar a Ontologia do agente e a linguagem SL0. Este registo é efectuado no próprio Agente, definindo assim quais as suas características ao nível de vocabulário conhecido e qual a forma como codifica as mensagens.

4.3 Módulo Interface Gráfica

Este módulo implementa a interface do sistema com o utilizador. A interface assenta numa janela principal que abre uma vez carregado o módulo mundo.

Através desta interface o utilizador pode construir um mundo, criando caminhos, agentes e colocando uma fonte ou pode carregar a estrutura dum mundo previamente

guardado. Para a criação dos agentes existe um componente do módulo exclusivamente dedicado a obter do utilizador as diversas propriedades do agente.

No módulo interface estão também implementados componentes que permitem comunicar com o módulo mundo, criando assim a possibilidade de ordenar uma execução do mundo, pausa-la quando for necessário e visualizar em tempo real a movimentação dos agentes.

4.4 Módulo Mundo

Este módulo é o primeiro a ser carregado quando se inicia a aplicação. É da sua responsabilidade iniciar a plataforma[2] e posteriormente a interface gráfica. Sendo este o módulo central da aplicação, tem implementado funcionalidades para comunicar com interface, ordenar à plataforma que crie agentes e responder a mensagens vindas dos agentes criados.

A comunicação com a interface tem como objectivo inicial obter uma estrutura para o mundo e posteriormente manter uma visualização das alterações do estado do mundo. A troca de mensagens, uma vez que é cíclica está implementada num *behaviour* especial.

Mais detalhes sobre a comunicação mundo-interface, mundo-plataforma e a forma como o mundo escuta as mensagens dos seus agentes são descritos na secção Detalhes da Implementação.

4.5 Módulo Behaviours

Este módulo engloba todos os comportamentos de cada um dos tipos de agentes da aplicação. Existem fundamentalmente dois tipos de agentes distintos, os agentes com arquitectura BDI e o mundo.

- ***AgenteBehaviour*** Impõe que um agente com este tipo de comportamento siga rigorosamente todos os passos da arquitectura BDI apresentada na figura [2]. Permite também ao agente comunicar com o mundo.

[2] Existe também a possibilidade de primeiramente iniciar a plataforma e a partir desta iniciar o mundo.

- ***MundoBehaviour*** Caracteriza o comportamento do mundo. O seu único objectivo é estar constantemente a escutar mensagens vindas dos agentes BDI. A troca de mensagens entre o mundo e os agentes BDI é feita recorrendo a protocolos especificados pela FIPA [1] conforme descrito na secção Detalhes da Implementação.

5 Esquemas de Representação de Conhecimento e Metodologia

Esta secção pretende especificar os esquemas de representação de conhecimento utilizados. É também feita a descrição processos de raciocínio dos agentes e da comunicação entre agentes.

5.1 Estrutura dos Agentes BDI

Este trabalho implementa agentes baseados na Estrutura BDI. Esta estrutura apresenta-se na figura 2[3] e é utilizada nos três tipos de agentes existentes. De seguida é feita a descrição da figura.

Um agente BDI começa por obter as percepções, que representam o conjunto de objectos que em determinada altura um agente visualiza no mundo. Com base nestas percepções são construídas as crenças, as crenças existentes também são incluídas. Com base nas crenças são gerados os desejos, que representam os objectivos que o agente pode satisfazer em determinada altura. As intenções representam a forma como o agente pode satisfazer cada desejo, estas são geradas de acordo com as crenças e os desejos. O passo seguinte possibilita gerar as acções que uma agente pode efectuar e pela selecção da que melhor se adequa as intenções obtidas. Finalmente procede-se à execução da acção. Posteriormente voltaremos a detalhar esta estrutura.

[3] Baseada em, acetatos de apoio as aulas de AIAD, Prof. Eugénio Oliveira
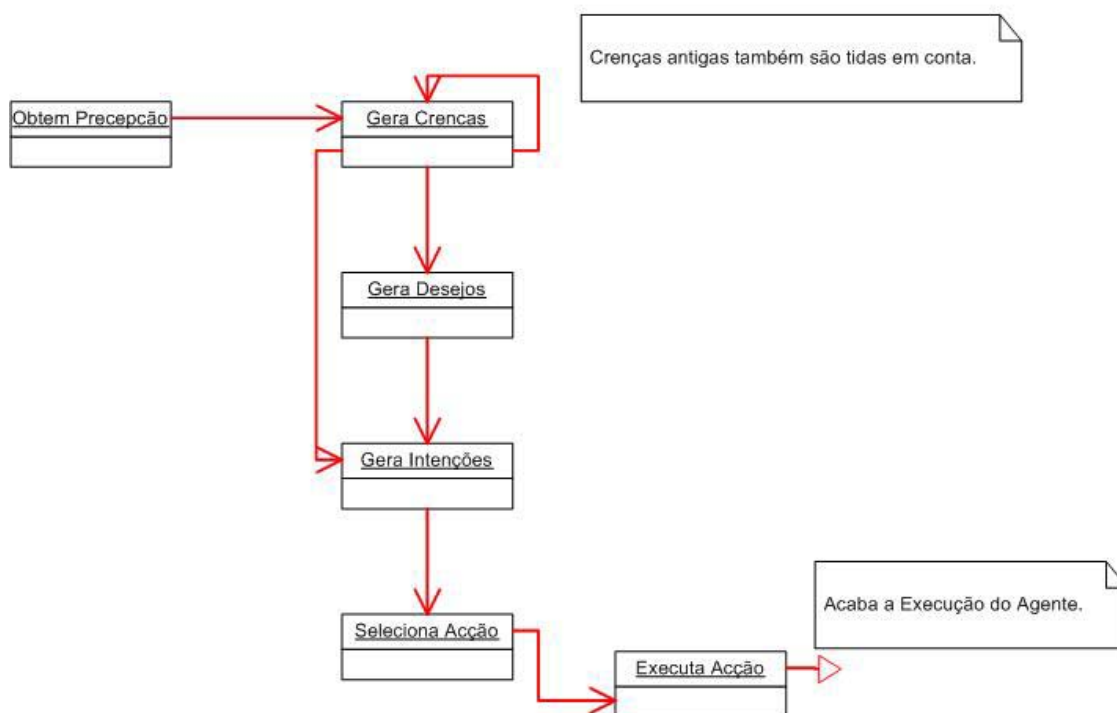
Figura 2 Estrutura dos Agentes BDI implementados.

5.2 Estrutura interna do Mundo no Agente BDI

Como já foi dito cada agente guarda internamente a estrutura do mundo. Dado que o agente não conhece o mundo completamente, devido ao seu raio de visão, esta estrutura interna tem de ser actualizada em cada iteração do agente, com base nas percepções recebidas. Desta forma cada agente vai construindo um grafo na sua memória.

Optou-se por esta representação para assim ser possível ao agente deslocar-se em todas as direcções sem perder informação sobre aquilo que já viu. A implementação em grafo foi aqui adoptada pois possibilita a adopção de algoritmos já conhecidos para encontrar caminhos entre dois nós, nomeadamente o algoritmo de Djikstra.

5.3 Estrutura do Agente Mundo

O agente mundo tem uma estrutura não inteligente, este limita-se apenas a criar os agentes e a responder aos pedidos efectuados por estes. Este agente tem uma representação de si próprio através do conceito "CMundo", presente na ontologia. Este conceito especifica os caminhos (conceito "Caminhos") existentes a fonte (conceito "Fonte") e os agentes (conceito "AgenteConceito") que estão presentes no mundo. Quando um agente toma uma decisão, comunica-a ao agente mundo para que este possa proceder à actualização do seu mundo. Seguidamente este agente procede à actualização da interface gráfica.

5.4 Comunicação entre Agentes

Como já foi referido, neste sistema existe comunicação entre os agentes BDI e o agente mundo. Esta comunicação permite ao agente obter as suas percepções e no fim permite ao mundo saber qual a acção que o agente executou para assim poder actualizar a interface gráfica.

A plataforma Jade disponibiliza uma funcionalidade, Sniffer, que permite visualizar a troca de mensagens entre os agentes. Na figura 3 é visível um exemplo, o qual se passa a explicar. Cada mensagem é representada por uma linha.

O Sniffer foi associado ao agente mundo, possibilitando assim visualizar todas as mensagens de interacção com ele. No sistema foi introduzido um agente BDI, representado na 3 pelo nome "agente1".

A primeira mensagem (0) é do tipo INFORM e representa a indicação que o mundo dá ao agente, dos seus parâmetros iniciais. Ou seja esta mensagem é constituída por um conceito "AgenteConceito", que está incluído na Ontologia. Salienta-se aqui que é o agente mundo quem "lança" os restantes agentes e depois lhes manda a esta mensagem com os seus parâmetros iniciais.

Seguidamente o agente responde com uma mensagem do tipo QUERY-REF, esta mensagem representa o pedido por parte do agente das suas percepções. Esta mensagem é constituída por um conceito "ObtemPrecepcoes", incluído na ontologia.

A mensagem número 2 significa a mensagem de confirmação, AGREE, que o mundo manda ao agente, significa que seguidamente lhe vai passar a informação pedida.

Esta informação é o conteúdo da mensagem número 3, que é uma "percepcao". Esta é uma mensagem de preformativa INFORM.

A mensagem número 4, de preformativa REQUEST, representa o pedido que o agente faz ao mundo para que este actualiza a interface gráfica. Este pedido é feito após o agente ter tomado a decisão sobre a acção a tomar. Seguidamente o mundo manda ao agente uma mensagem de preformativa AGREE, que significa que o mundo vai executar o pedido. Esta mensagem é seguida pela mensagem número 6 de preformativa INFORM que indica ao agente a execução do pedido.

A mensagem número 7 é um novo pedido de percepções, esta mensagem dá início a um novo ciclo semelhante ao definido pelas mensagens 1 a 6.
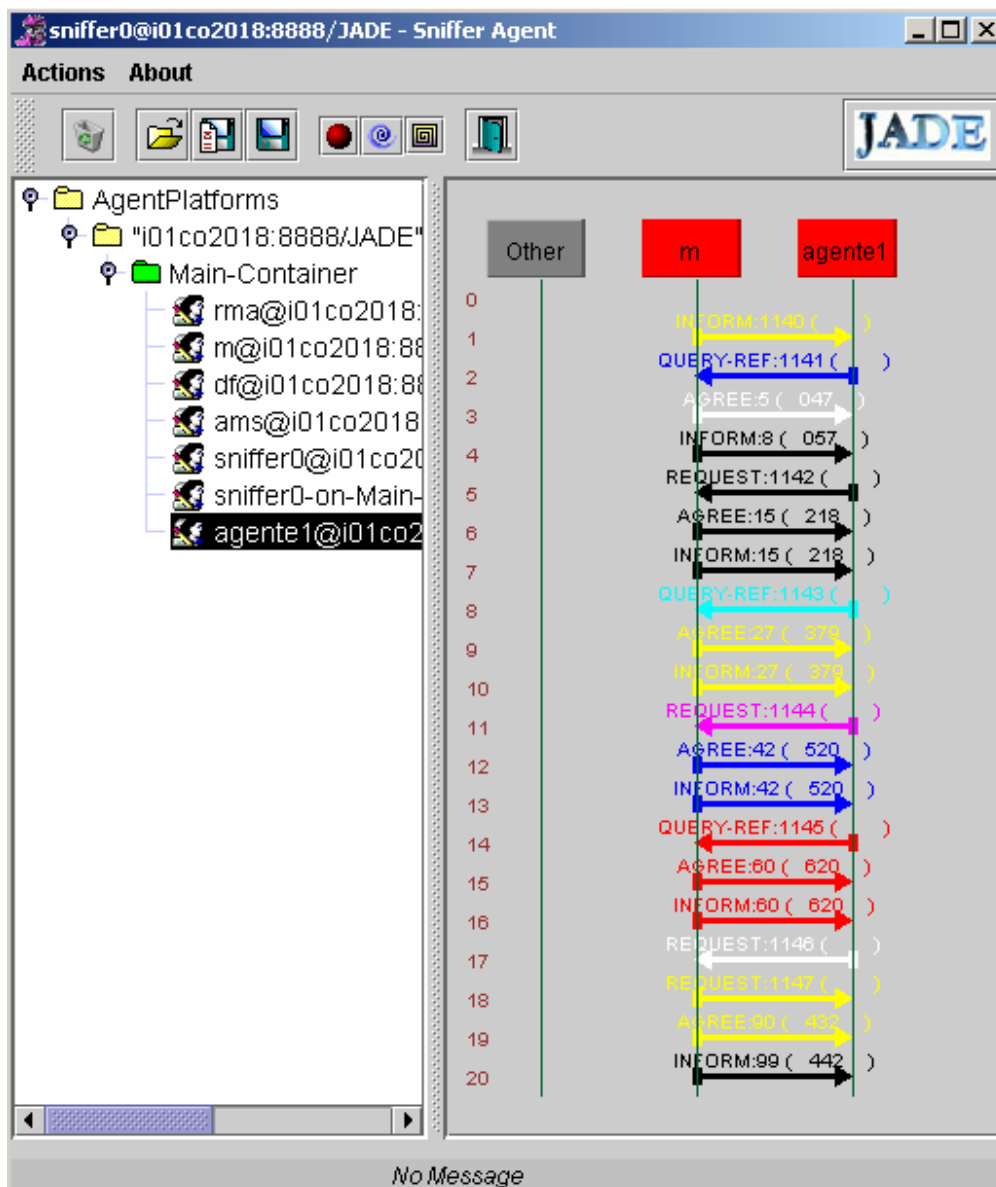
Figura 3 Exemplo de uma execução da comunicação entre agentes, usando a funcionalidade Sniffer.

6 Detalhes de Implementação

Toda aplicação assenta na plataforma JADE. Esta fornece as bases para a comunicação entre agentes e execução concorrente de agentes, um agente por *thread*. Dentro dum agente a plataforma também se encarrega de executar vários comportamentos de forma concorrente [5].

6.1 Detalhe do agente BDI

O agente inicializa a execução de um ciclo, pedindo ao mundo as percepções. Estas representam aquilo que esse agente está a visualizar nesse momento, ou seja tudo o que está no seu raio de visualização. Uma percepção tem a seguinte estrutura:

Vector agentesBom;
Vector caminhos;
Vector agentesEnergia;
Vector agentesMonstro;
Fonte fonte;

Vector caminhos Inclui todos os caminhos que o agente vê, salienta-se aqui que apenas são incluídos os caminhos, dos quais é visível a origem e o destino, ou seja o caminho em toda a sua existência.
agentesBom Inclui todos os agentes Bons que estão no raio de visão do agente.
agentesEnergia Inclui todos os agentes Energia que estão no raio de visão do agente.
agentesMonstro Inclui todos os agentes Monstro que estão no raio de visão do agente.
fonte Inclui a posição da Fonte caso esta seja visível pelo agente.

Seguidamente o agente executa uma das suas funcionalidades que permite gerar as crenças. Esta funcionalidade começa por pegar nas crenças antigas e incrementa-lhes a idade. Existe um valor pré definido que especifica a validade das crenças, desta forma crenças que atinjam um valor de idade superior à validade, deixam de ser consideradas. Existe uma excepção ao que foi dito, esta prende-se com a fonte, dado que esta é estática, uma crença na sua existência num determinado sitio é sempre válida. Seguidamente a este processo são criadas as novas crenças, com idade zero. A estrutura de uma crença é a seguinte:

String quem;
No onde;

int quando;

quem Representa em que se baseia a crença.
onde Representa a posição onde "quem" existe ou existiu.
quando Representa a idade da crença.

Seguidamente vão ser apresentadas as crenças existentes (possíveis no campo quem) no programa e que são comuns a todos os agentes.

CRENCA_AGTBOM Representa uma crença na existência de um agente bom.
CRENCA_AGTMONSTRO Representa uma crença na existência de um agente monstro.
CRENCA_AGTENERGIA Representa uma crença na existência de um determinado agente energia.
CRENCA_FONTE Representa uma crença que se baseia na existência da fonte.

Após este processo de geração de crenças, os agentes executam a geração de desejos, estes têm como base as crenças que o agente tem. Seguidamente apresenta-se a estrutura de um desejo.

String nome;

nome Representa a descrição do desejo.

De seguida apresentam-se os tipos de desejos que cada agente pode ter.

Agente Bom:
DESEJO_FUGIR O agente deseja fugir porque conhece a posição de um agente monstro (existência de uma crença no monstro com idade zero).
DESEJO_ENERGIA O agente deseja obter energia porque sabe que existe um agente energia num dado ponto (existência de uma crença no monstro com idade zero).
DESEJO_FONTE O agente deseja alcançar a fonte pois sabe a sua posição.

DESEJO_EVITAR O agente deseja evitar uma determinada posição pois sabe que lá viu um agente monstro anteriormente (existência de uma crença no monstro com idade superior a zero)
DESEJO_ALCANCAR O agente deseja alcançar uma determinada posição pois sabe que lá viu um agente energia anteriormente (existência de uma crença no agente energia com idade superior a zero)

Agente Monstro:
DESEJO_ENERGIA O agente deseja obter energia porque sabe que existe um agente energia num dado ponto (existência de uma crença no monstro com idade zero).
DESEJO_APANHAR O agente deseja apanhar porque conhece a posição de um agente bom (existência de uma crença no agente bom com idade zero).
DESEJO_ALCANCAR_BONS O agente deseja alcançar uma determinada posição pois sabe que lá viu um agente bom anteriormente (existência de uma crença no agente bom com idade superior a zero).
DESEJO_ESTAR O agente viu a fonte e deseja estar nas suas imediações pois sabe que a ela tenderam a convergir os agentes bons.
DESEJO_ALCANCAR_ENERGIA O agente deseja alcançar uma determinada posição pois sabe que lá viu um agente energia anteriormente (existência de uma crença no agente energia com idade superior a zero).

Agente Energia:
DESEJO_APANHAR O agente deseja alcançar uma agente (bom ou monstro) pois eles necessitam de aumentar a sua energia e desta forma o agente energia aumenta a sua alegria (existência de uma crença na existência de agentes com idade igual a zero).
DESEJO_ALCANCAR APANHAR O agente deseja alcançar uma agente (bom ou monstro) pois eles necessitam de aumentar a sua energia e desta forma o agente energia aumenta a sua alegria (existência de uma crença na existência de agentes com idade superior a zero).
DESEJO_ESTAR O agente viu a fonte e deseja estar nas suas imediações pois sabe que a ela tenderam a convergir os agentes (bons e monstros).

Após este processo de obtenção dos desejos, o agente cria as suas intenções. Estas são geradas a partir das crenças e dos desejos.

String nome
String porque
Vector parametros

nome representa a definição da intenção.
porque representa o motivo que leva a que um agente opte por cwerta intenção.
parametros guarda parametros que serão necessário quando se tiver de optar por uma acção, nomeadamente a posição e a idade da crença que dão origem à intenção.

Seguidamente apresentam-se as várias intenções possíveis que podem ser geradas para cada agente.

Agente Bom:
INTENCAO_IR Intenção de ir para uma determinada posição.
INTENCAO_FUGIR Intenção de fugir de uma determinada posição.
INTENCAO_EVITAR Intenção de evitar uma determinada posição.
INTENCAO_ALCANCAR Intenção de alcançar uma determinada posição.

Motivos

PORQUE_MONSTRO O motivo pelo qual o agente deseja fazer algo é a existência de um agente monstro.

PORQUE_FONTE O motivo pelo qual o agente deseja fazer algo é a existência da fonte.

PORQUE_ENERGIA O motivo pelo qual o agente deseja fazer algo é a existência de um agente energia.

PORQUE_POSSIVEL_MONSTRO O motivo pelo qual o agente deseja fazer algo é a possível existência de um agente monstro.

PORQUE_POSSIVEL_ENERGIA O motivo pelo qual o agente deseja fazer algo é a possível existência de um agente energia.

Agente Monstro

INTENCAO_IR Intenção de ir para uma determinada posição.
INTENCAO_ALCANÇAR Intenção de alcançar uma determinada posição.
INTENCAO_ESTAR Intenção de estar numa zona circundante à fonte.
INTENCAO_APANHAR Intenção de apanhar um agente bom que se encontra em determinada posição.

Motivos

PORQUE_BOM O motivo pelo qual o agente deseja fazer algo é a existência de um agente bom.

PORQUE_FONTE O motivo pelo qual o agente deseja fazer algo é a existência da fonte.

PORQUE_ENERGIA O motivo pelo qual o agente deseja fazer algo é a existência de um agente energia.

PORQUE_POSSIVEL_BOM O motivo pelo qual o agente deseja fazer algo é a possível existência de um agente bom.

PORQUE_POSSIVEL_ENERGIA O motivo pelo qual o agente deseja fazer algo é a possível existência de um agente energia.

Agente Energia

INTENCAO_IR Intenção de ir para uma determinada posição.
INTENCAO_ALCANCAR Intenção de alcançar uma determinada posição.
INTENCAO_ESTAR Intenção de estar numa zona circundante à fonte.

Motivos

PORQUE_AGENTE O motivo pelo qual o agente deseja tomar certa atitude, deve-se a existência de um agente (bom ou monstro) numa determinada posição.

PORQUE_POSSIVEL_AGENTE O motivo pelo qual o agente deseja tomar certa atitude, deve-se à possível existência de um agente (bom ou monstro) numa determinada posição.

PORQUE_FONTE O motivo que leva um agente a tomar certa atitude, é a existências da fonte numa de terminada posição.

O passo seguinte permite ao agente o seleccionar a acção que vai executar. Para seleccionar a acção, o agente começa por obter todas as acções que pode realizar, estas consistem em deslocar-se para um nó adjacente ou por ficar parado.

A selecção da acção é feita tendo em conta um conjunto de factores numéricos. Desta forma, o processo de selecção das acções consiste em, identificar todas as intenções, com base nessas identificações, são seleccionadas as acções que possibilitam satisfazer as intenções. Finalmente para cada uma destas relações é calculado um factor numérico que fica associado a cada acção. No final calcula-se a acção que tem o maior somatório de factores numéricos e essa é a seleccionada. Os factores numéricos têm em conta o tipo de intenção, a distancia ao motivo da intenção e a idade da crença que fundamenta a intenção. Esse factor numérico tem ainda em conta o valor de "aventureirismo" de cada agente. Ou seja um agente aventureiro arrisca mais em certo tipo de acções que outros. De seguida é descrita a estrutura de uma acção.

String accao;
No onde;
Vector factores;

accao Determina a acção possível que um determinado agente pode ter. Para os três tipos de agentes, existem dois tipos de acção:

ACCAO_ANDAR Uma acção pode ser andar para uma determinada posição.

ACCAO_PARADO Uma acção pode ser ficar parado.

quem Determina a posição associada à acção. Ou seja se for uma acção de andar, diz para onde e se for parado especifica a posição actual.

factores Local onde se guardam os factores numéricos que permitem seleccionar a melhor acção.

No final o agente executa a acção, ou seja se a acção seleccionada for uma acção de andar, então altera a sua posição, se for uma acção de ficar parado, não faz nada.

O ciclo de execução de um agente inicializa-se com o pedido ao mundo das percepções. No final um agente comunica ao mundo qual a acção que este proceda à actualização gráfica.

6.2 Detalhes da implementação do Mundo

O mundo é implementado na classe *Mundo.java* que é uma extensão à classe *GUIAgent* (que faz parte da *Framework* JADE), este é um agente da plataforma que tem a particularidade de receber eventos vindos duma interface gráfica executando num *thread* diferente.

Como todos os agentes JADE também o mundo implementa o método *setup*. Este método é chamado pelo *agent container* aquando da criação do agente e faz o registo da ontologia, linguagem, e dos seus *behaviours*. Ver figura 4.

```
...
// Regista o codec para a linguagem SL0
getContentManager().registerLanguage(new SLCodec(),
FIPANames.ContentLanguage.FIPA_SL0);

// Regista a ontologia usada pela aplicação
getContentManager().registerOntology(Ontologia.getInstance());

//lança a interface gráfica
setupGUI();

// comportamento para ler mensagens
h = ReceiverBehaviour.newHandle();
this.addBehaviour(new ReceiverBehaviour(this, h, 100000));

// comportamento para excutar alterações ao estado do mundo
this.addBehaviour(new MundoBehaviour(this));
...
```

Figura 4 Excerto do método *setup* do mundo.

Outra função importante, referida na descrição dos módulos, implementada nesta classe é a capacidade de arrancar a plataforma a partir da execução desta classe. Esta funcionalidade é assegurada no método estático *main*.

```
public static void main(String args[])

	(...)
	Profile pMain = new ProfileImpl(null, 8888, null);
	MainContainer mc = rt.createMainContainer(pMain);

	AgentController rma = mc.createNewAgent("rma",
"jade.tools.rma.rma", new Object[0]);

	//inicia o agente visualizador da plataforma
	rma.start();

	AgentController mundo = mc.createNewAgent( "m",
"theGame.Mundo",null);
	// inicia o mundo (agente)
	mundo.start();
	(...)
```

Figura 5 Excerto do método *main* do mundo.

O método responsável pela leitura dos eventos enviados pela interface é o método *onGuiEvent(GuiEvent ev)*. O argumento *ev* instância da classe *GuiEvent* trás informação sobre o tipo do evento (usado para diferenciar os diferentes comandos vindos da interface gráfica) e uma lista de parâmetros que cada comando requer. Em seguida está um extracto da função onde é recebido um pedido de execução do mundo.

```
protected void onGuiEvent(GuiEvent ev)

        (...)
        switch(ev.getType()) {
            case INICIA:
                m = (CMundo)ev.getParameter(0);

                this.mundo.setFonte(m.getFonte());
                this.mundo.setCaminhos(m.getCaminhos());
                this.mundo.setAltura(m.getAltura());
                this.mundo.setLargura(m.getLargura());
                this.mundo.setNos(m.getNos());

                for(i = 0; i < m.getAgentes().size();i++) {
                    ag = (AgenteConceito)m.getAgentes().elementAt(i);
                    this.registaAgente(ag);
                }
```

```
                break;
        (...)
```

Figura 6 Excerto do método *onGuiEvent* do mundo.

O último método que importa salientar na classe *Mundo.java* é o método *registaAgente(AgenteConceito ag)*. Recebendo informação referente a um agente o método cria um agente na plataforma, inicia-o e envia-lhe uma mensagem com as suas propriedades (contidas na variável *ag*).

```
public void registaAgente(AgenteConceito ag)

            (...)
            if (ag instanceof AgenteBom) {
                guest = container.createNewAgent(localName,
"theGame.agentes.AgBom", null);
                // inicia a execução do agente num thread separado
                guest.start();
            }
            (...)
            ag.setMundo(mundo);
            ag.setId(new AID(localName, AID.ISLOCALNAME));
            mundo.addAgente(ag);
            (...)
            msg.setPerformative(ACLMessage.INFORM);
            msg.setOntology(Ontologia.NAME);
            msg.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
            msg.setContentObject(ag);
            msg.addReceiver(ag.getId());

            send(msg);
            (...)
```

Figura 7 Excerto do método *onGuiEvent* do mundo

6.3 Detalhes da implementação da interface gráfica

A interface gráfica está implementada na classe ***MundoGUI.java***. Esta classe é derivada da classe da *Framework JDK JFrame.java*. Tem no entanto um construtor diferente que recebe uma referência para um objecto instância da classe *Mundo.java*. Este será utilizado para enviar eventos ao mundo (que está a correr num *thread* diferente).

```
private void iniciarActionPerformed(java.awt.event.ActionEvent evt)
    {
      if (mundo != null)
            {
```

```
                if (this.cmundo.getAgentes().size() > 0)
                {
                      GuiEvent ev = new GuiEvent(null,mundo.INICIA);
                      ev.addParameter(this.cmundo);
                      mundo.postGuiEvent(ev);
                }
          }
    }
```

Figura 8 –método *iniciarActionPerformed.*

Na figura 8 é possível ver o método que ordena ao mundo que inicie a sua execução, passando-lhe como parâmetro um objecto com a representação do mundo tal como especificado na ontologia.

6.4 Detalhes da classe AgenteBehaviour.java

Esta classe define o comportamento de todos os agentes com arquitectura BDI. A arquitectura BDI é bastante complexa e como a plataforma JADE não a suporta directamente houve necessidade de adaptar um dos *behaviours* da plataforma, o *behaviour CyclicBehaviour*. Um agente com este comportamento está num de seis estados possíveis:

- INICIAL – espera pela mensagem com as propriedades iniciais do agente.
- ENVIA_GET_PERC – Envia mensagem ao mundo com pedido de consulta das percepções do agente.
- ESPERA_PERC – Este estado espera uma resposta ao pedido de consulta das percepções do agente.
- CICLO_BDI – Executa o ciclo característico da arquitectura BDI. Ver figura 2.
- ENVIA_ACTUALIZA – Envia mensagem ao mundo com pedido de actualização da interface gráfica.
- ESPERA_ACTUALIZA – Espera por uma resposta ao pedido de actualização da interface gráfica.
- DORMINDO – Espera que lhe seja enviada uma mensagem com pedido de "Acordar".

Para uma melhor compreensão da forma como funciona um agente com este comportamento apresenta-se em seguida o diagrama de estados.
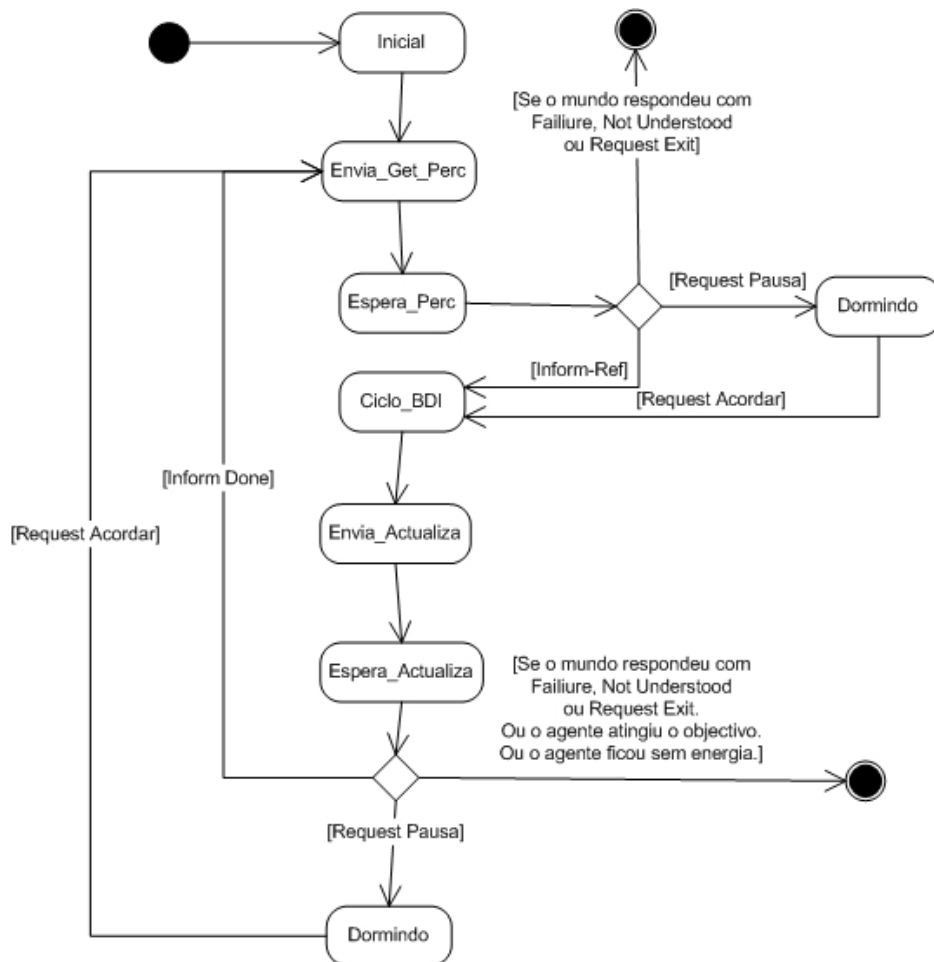
Figura 8 – diagrama de estados do comportamento dum agente BDI.

6.5 Detalhes da classe MundoBehaviour.java

Esta classe define o comportamento do mundo. A função principal do comportamento é esperar mensagens vindas dos agentes sem usar *busy wating* e despacha-las. Para conseguir isso recorre-se a um *behaviour* disponibilizado na *framework* JADE denominado *ReceiverBehaviour*. Como este *behaviour* tem apenas como finalidade esperar por **uma** mensagem sem usar *busy wating*, foi necessário usar outro *behaviour* para tornar cíclico o processo de espera de mensagens. O *behaviour*

usado está implementado na classe *MundoBehaviour.java* e entende a classe da *framework* JADE *CiclicBehaviour.*

A troca de mensagens envolvendo a consulta das percepções processa-se respeitando o protocolo FIPA_QUERY [9]. A troca de mensagens relativas ao pedido de actualização da interface segue o protocolo FIPA_REQUEST [9].

A plataforma JADE tem *behaviours* que implementam os protocolos mencionados, mas porque não houve sucesso no uso dessas funcionalidades implementamos esses protocolos nesta classe.
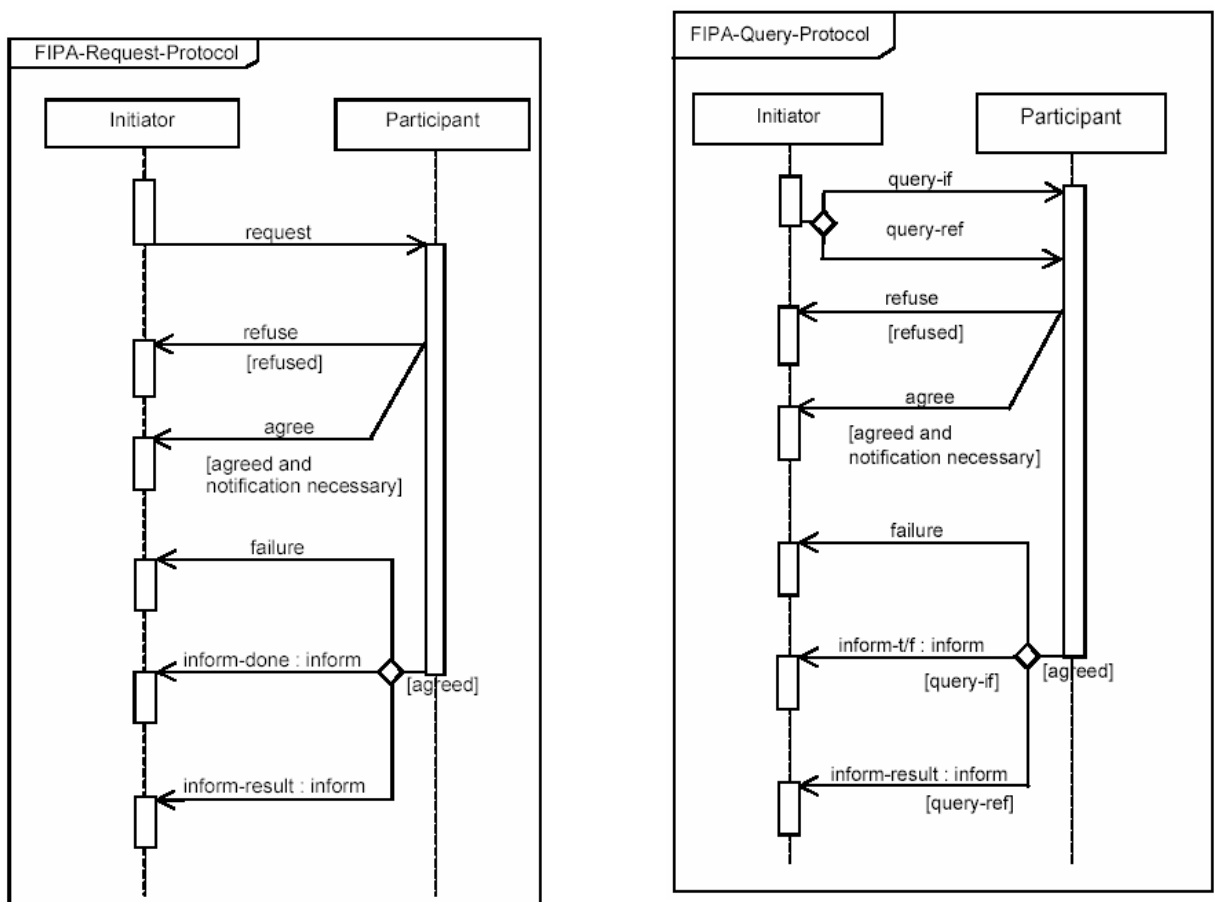
Figura 9 Protocolos FIPA REQUEST PROTOCOL e FIPA QUERY PROTOCOL.

6.6 Detalhes de Implementação das funções de actualização de valores

Cada agente tem vários atributos como a alegria (relativo a todos), energia ("Bom" e "Monstro"), medo ("Bom") e cólera ("Monstro"). Aos valores definidos inicialmente na criação dos agentes, são aplicados métodos que os actualizam; estes métodos aplicam certas regras para decrescer ou aumentar esses valores, apesar de implementados, apenas o método que actualiza a energia está a ser utilizado..

ActualizaAlegria Método que actualiza a alegria dos agentes. Para o agente "Bom", o método recebe o agente a actualizar e um vector de caminhos que levam a um monstro. O método calcula, de entre aqueles caminhos, o mais curto e baseado nessa distância utiliza uma função que aumenta se a distância for superior a um determinado valor e decresce se for inferior a esse mesmo valor. Para o agente monstro o método é semelhante diferindo em que o vector é o dos caminhos que levam a um agente "Bom" e que para decrescer a alegria, a distância ao agente "Bom" tem de ser superior a um certo valor e para aumentar essa distância tem de ser inferior a esse mesmo valor. Para o agente "Energia" apenas é usado um valor *booleano* para verificar se numa iteração o agente forneceu ou não energia a uma agente e se sim a sua alegria é aumentada, em caso contrário esta é decrescida.

ActualizarEnergia método que actualiza a energia dos agentes. É um método igual para o agente "Bom" e "Monstro". O método recebe o agente que vai ser actualizado e uma posição actual do agente e outra para onde se irá deslocar. A distância entre os dois nós é calculada, formando um peso que é utilizado numa função para calcular quanta energia é gasta nesse movimento.

ActualizarMedo método que actualiza o medo de determinado agente "Bom". Sendo um valor que varia inversamente em relação à alegria, a função calculada é a mesma mas enquanto a alegria varia directamente com a distância, o medo varia inversamente com a distância.

AcualizarCólera método que actualiza a cólera de determinado agente "Monstro". Sendo um valor que varia inversamente em relação à alegria, a função

calculada é a mesma mas enquanto a alegria varia inversamente com a distância, a cólera varia directamente com a distância.

7 Ambiente de desenvolvimento

O nosso projecto foi desenvolvido utilizando a linguagem JAVA usando, como IDE, o *JCreator Pro* da *Xinox Software* e JDK 1.4 como livrarias de suporte à linguagem Java.

O trabalho foi realizado utilizando os recursos informáticos disponibilizados pela faculdade. As características dos computadores empregues para o desenvolvimento de todo o código da aplicação são:

- Sistema Operativo Windows 2000 Professional.
- Intel Pentium 733 MHz com 512 ou 128 Mb de memória ram.

A plataforma utilizada foi o JADE.

8 Conclusão

Uma das maiores dificuldades que surgiram durante a realização deste projecto prende-se com a plataforma Jade, nomeadamente por esta não auxiliar a criação de agentes BDI, desta forma toda a estrutura destes agentes teve de ser desenvolvida.

Verificou-se que um dos *behaviours*, OneShotBehaviour, não funciona de acordo com o descrito na API. Esta deficiência atrasou bastante o desenrolar do trabalho, uma vez que só depois de termos a estrutura do *behaviourBDI* feita é que detectamos o problema, cuja resolução passou por alterar a estrutura.

Do que nos propusemos a fazer ficou por fazer a actualização de algumas características dos agentes, apesar de os métodos estarem implementados, não estão a ser chamados.

Tivemos dificuldades em registar agentes de diferentes tipos no mesmo mundo. Apesar de os agentes serem equivalentes e de conseguirmos ter vários agentes do mesmo

tipo a correr no mundo, verificou-se que quando incluíamos agentes de diferentes tipos a plataforma bloqueava no registo dos agentes.

9 Melhoramentos

O tema proposto neste trabalho é bastante rico em possíveis melhoramentos, pois é um tema que permite a utilização da imaginação para o aperfeiçoamento, do comportamento dos agentes. Existem muitas formas que podiam ser usadas para implementar a "inteligência" do agente, nomeadamente recorrendo a métodos de Inteligência Artificial. Desta forma os melhoramentos mais importantes deveriam estar relacionados com o aperfeiçoamento dos agentes BDI, nomeadamente na forma como este procede à selecção da acção.

Um melhoramento óbvio é o da interface gráfica. Este não era o principal objectivo deste trabalho, por isso a interface aqui apresentada é bastante simples e não implementa algumas funcionalidades que seriam importantes, nomeadamente a possibilidade de o utilizador incluir alterações ao mundo (apenas pelo acréscimo de caminhos e agentes) após o os agentes estarem a executar.

Este trabalho não inclui cooperação entre os agentes, desta forma seria interessante implementá-la. Essa cooperação seria feita entre agentes do mesmo tipo. Por exemplo se existissem vários agentes bons, cada agente que visse um agente monstro comunicaria-o aos restantes agentes bons, passando esta informação a fazer parte das crenças do agente que as recebe-se. Processos semelhantes seriam implementados nos restantes agentes. Com uma estrutura deste tipo, poderia se complicar o sistema mais um pouco, introduzindo um factor de crença entre os agentes. Ou seja um agente que recebe uma informação, de um determinado agente do mesmo tipo, avalia se acredita muito ou pouco nesse agente. Essa avaliação poderia ser feita com base na importância da informação recebida anteriormente desse agente, na tomada das decisões.

Uma funcionalidade interessante, seria a hipótese de o utilizador incluir mais do que uma fonte, desta forma complicar-se-ia a decisão do agente.

10 Bibliografia

[1]www.fipa.org – Sítio *web* da FIPA

[2]http://www.fe.up.pt/~eol/PLADISMA/ARTIGOS/FIPA/ACL.html – FIPA 97 Specification, Version 2.0 Part 2 Agent Communication Language.

[3] http://sharon.cselt.it/projects/jade/ – Esta é a página oficial da plataforma JADE, onde é possível encontrar muita documentação, incluindo exemplos, tutoriais e o *javadoc* da API.

[4] JADE TUTORIAL-APPLICATION-DEFINED CONTENT LANGUAGES AND ONTOLOGIES, *Giovanni Caire,* incluído na documentação do Jade.

[5] JADE PROGRAMMER'S GUIDE, *Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimassa*, incluído na documentação do Jade.

[6] Acetatos de apoio às aulas de AIAD, Prof. Eugénio Oliveira, nomeadamente o capitulo 7 (acessíveis via *web* a partir de http://www.fe.up.pt/~eol/AIAD/aiad0203.html)

11 Glossário

Arquitecturas BDI - do inglês *"Believes, Desires and Intentions",* este tipo de arquitecturas têm em conta o conjunto definido por objectivos, crenças e intenções para definir as acções que o agente deve tomar. As crenças representam o estado actual enquanto os objectivos e desejos representam informação sobre o futuro a atingir.

Anexos

A - Manual do utilizador

Apresenta-se nesta secção um pequeno manual de utilização da plataforma gráfica desenvolvida.

Janela principal:
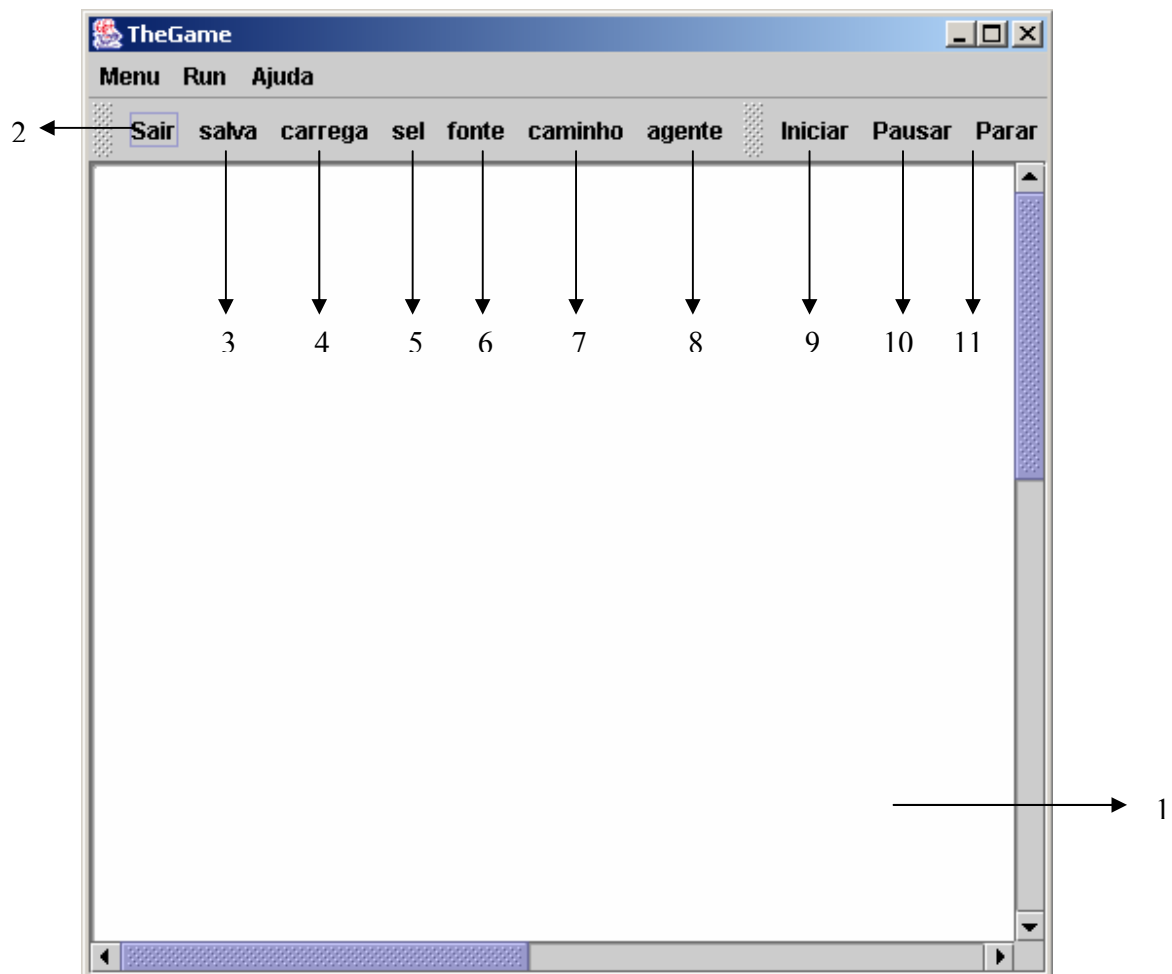
Figura 10 Janela principal.

Na figura 10 acima está representado a janela inicial sempre que é executada a aplicação

1. Área de desenho: Aqui são colocados, o mundo e os agentes.

2. Sair: Sai da aplicação.
3. Salva: Ao pressionar este botão, estando um mundo criado este é gravado num ficheiro para futuro uso.
4. Carrega: Carrega o mundo que foi previamente salvo.
5. Botão de selecção: permite a selecção de um agente ou caminho já criados podendo depois alterar as suas características. (não implementado)
6. Fonte: Coloca num dos nós uma fonte.
7. Caminho: cria um caminho através de um nó de início e de fim na área de desenho.
8. Agente: coloca um agente num nó já criado. (nova janela)
9. Início: corre a aplicação com o mundo e os agentes criados.
10. Pausa: Pressionado uma primeira vez depois de correr a aplicação, interrompe a execução, pressionado uma segunda vez resume a execução.
11. Parar: termina a execução. (não implementado)

Janela de criação de agentes
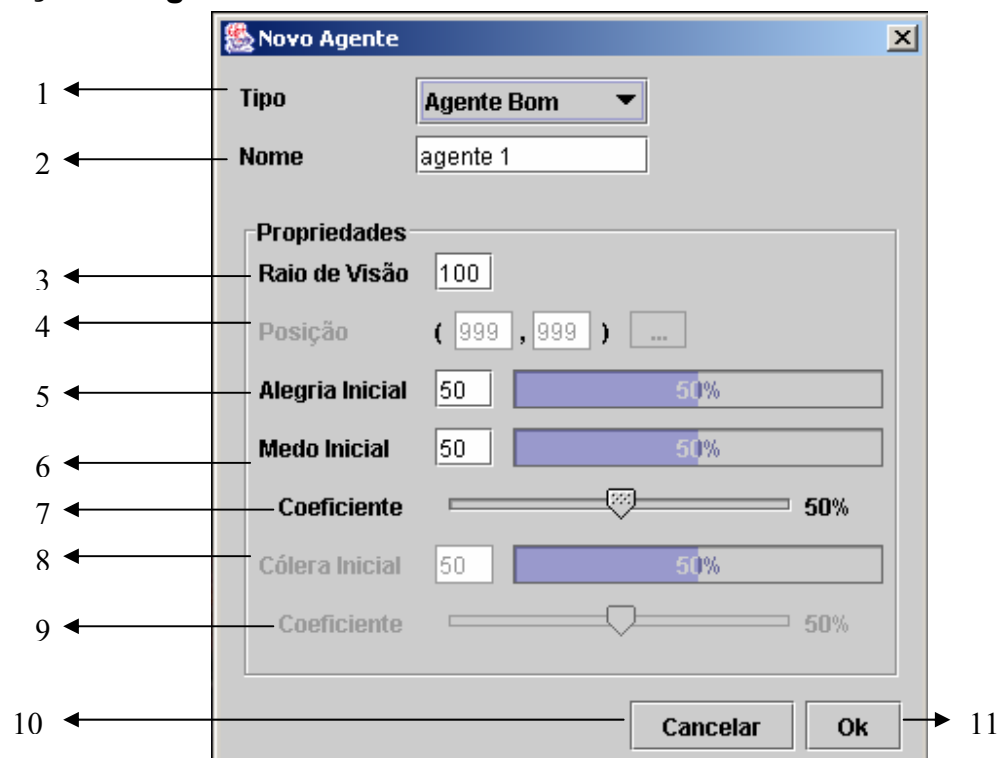
Figura 11 Janela de criação de agentes.

Na figura 11 acima está representada a janela que é apresentada ao utilizador sempre que um agente é colocado no mundo.

1. Tipo: Nesta caixa de texto define-se o tipo de agente que se vai introduzir.
2. Nome: Nesta caixa de texto insere-se o nome do agente a criar.
3. Raio de Visão: Valor que define o raio de visão de um agente.
4. Posição: número meramente informativo de onde se vai colocar o agente. (não implementado)
5. Alegria Inicial: Valor de alegria inicial do agente que pode variar entre 0 e 100
6. Medo Inicial: Activo apenas no caso do agente ser do tipo "Bom", este número define o valor do medo que o agente terá no início.
7. Coeficiente (Medo): Activa apenas no caso do agente ser do tipo "Bom", esta barra de deslizamento define o valor do coeficiente de medo do agente, que pode variar entre 0% e 100%.
8. Cólera Inicial: Activo apenas no caso do agente ser do tipo "Monstro", este número define o valor inicial de cólera que o agente terá.
9. Coeficiente (Cólera): Activa apenas no caso do agente ser do tipo "Monstro", esta barra de deslocamento define o valor do coeficiente de cólera do agente, que pode variar entre 0% e 100%.
10. Cancela: Cancela a colocação do agente.
11. Ok: Coloca o agente na área de desenho, no nó especificado, com os valores iniciais especificados.

B - Exemplo de uma execução

Apresenta-se de seguida uma sequência de *screenshots* durante a uma execução da aplicação:

Criação do grafo de caminhos
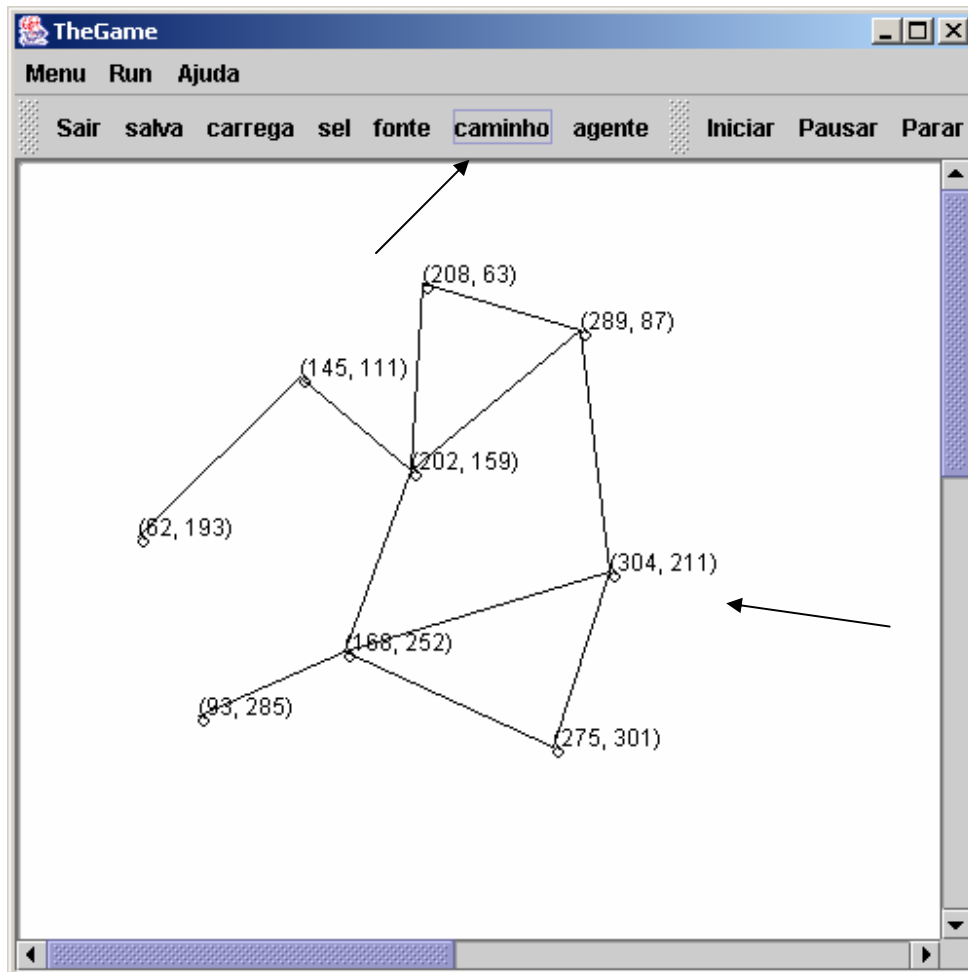
Figura 12 Criação de caminhos.

Através do botão "caminho" criam-se vários caminhos formando um grafo por onde os agentes se vão deslocar.

Colocação da fonte
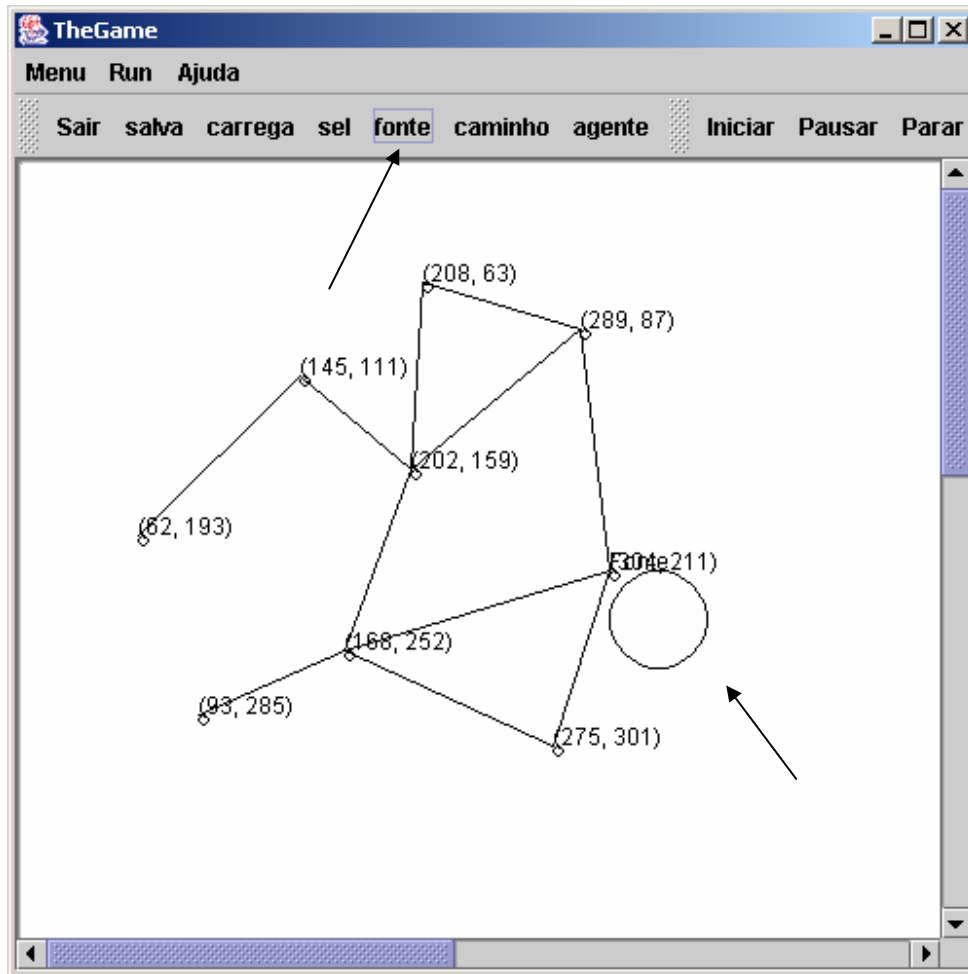
Figura 13 Inclusão da fonte.

Através do botão "fonte" coloca-se a fonte num dos nós disponíveis.

Colocação de agentes e definição de seus atributos

Atributos iniciais dos dois agentes colocados e os nós onde foram colocados.
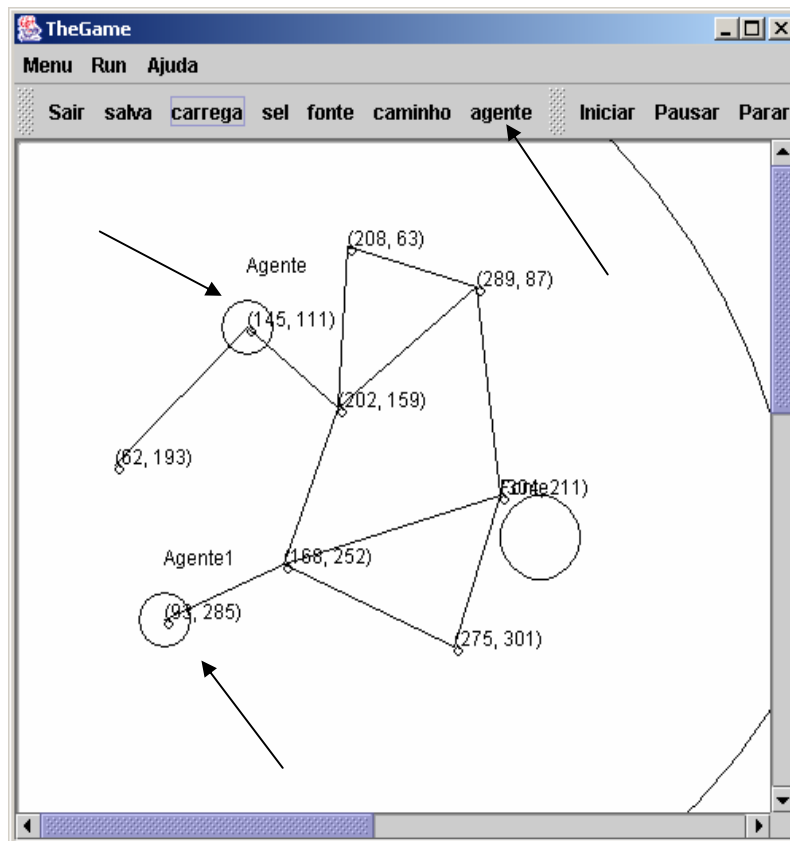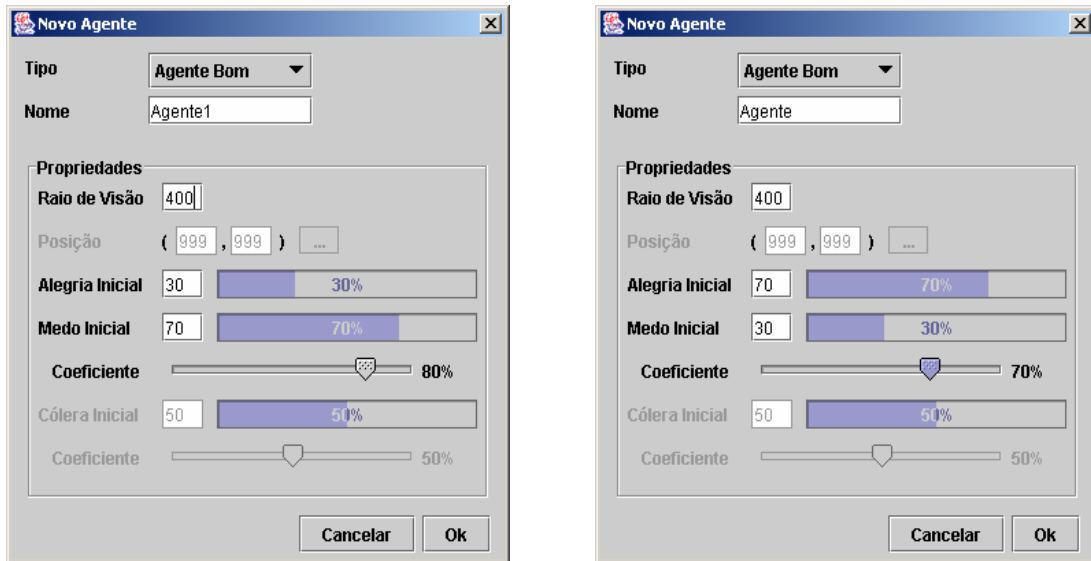
Figura 14 Colocação dos agentes no mundo.

Execução

Movimento dos agentes em direcção ao seu objectivo, a fonte
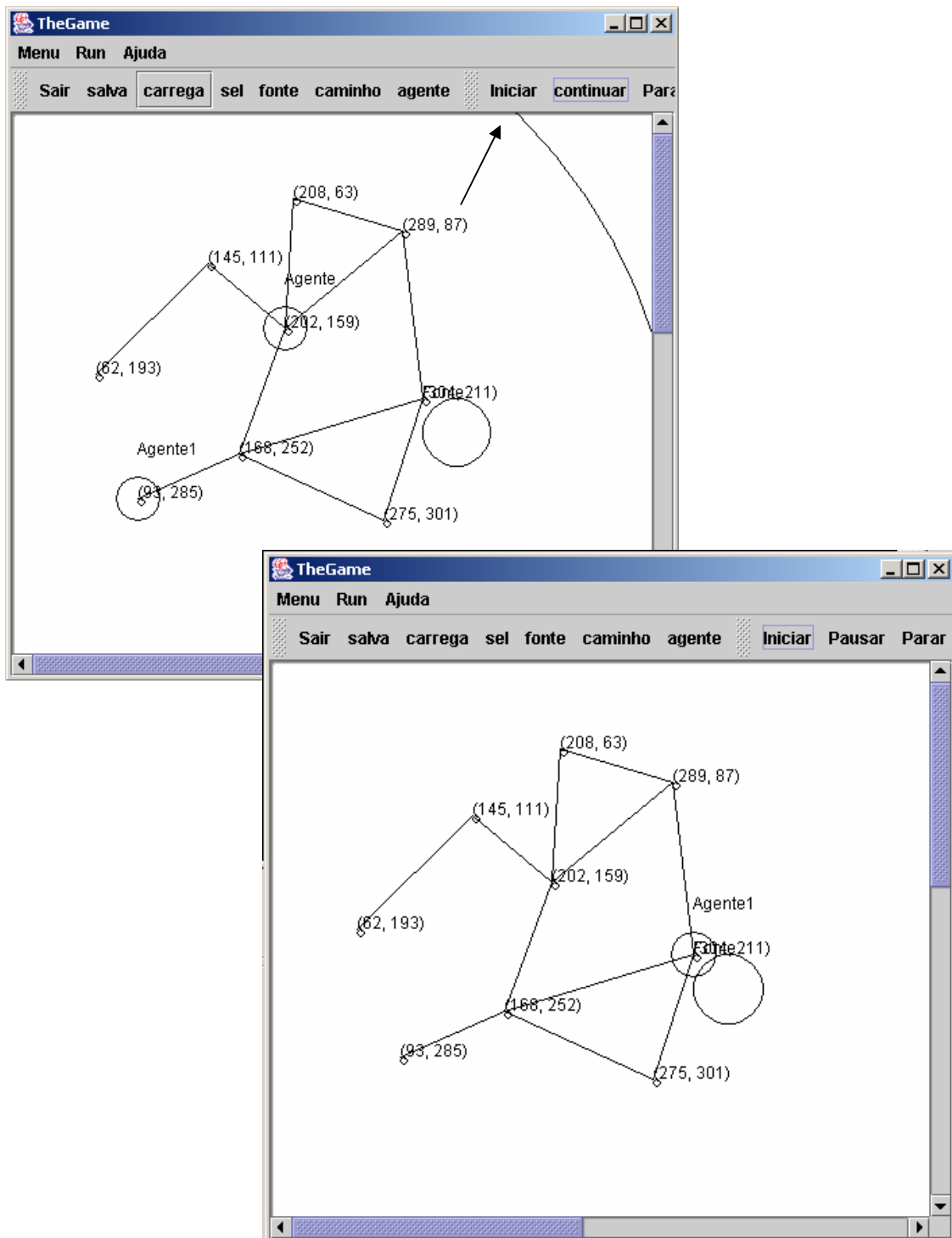
Figura 15 Sequência de execução do mundo.